LEARNING ROBUST DYNAMICS THROUGH VARIATIONAL SPARSE GATING

Arnav Kumar Jain[1,2*], **Shivakanth Sujit**[1,2], **Shruti Joshi**[1,2], **Vincent Michalski**[2,4]
Danijar Hafner[5,6], **Samira Ebrahimi-Kahou**[1,2,7]

ABSTRACT

Latent dynamics models learn an abstract representation of an environment based on collected experience. Such models are at the core of recent advances in model-based reinforcement learning. For example, world models can imagine unseen trajectories, potentially improving data-efficiency. Planning in the real-world requires agents to understand long-term dependencies between actions and events, and account for varying degree of changes, e.g. due to a change in background or viewpoint. Moreover, in a typical scene, only a subset of objects change their state. These changes are often quite sparse which suggests incorporating such an inductive bias in a dynamics model. In this work, we introduce the variational sparse gating mechanism, which enables an agent to sparsely update a latent dynamics model's state. We also present a simplified version, which unlike prior models, has a single stochastic recurrent state. Finally, we introduce a new ShapeHerd environment, in which an agent needs to push shapes into a goal area. This environment is partially-observable and requires models to remember the previously observed objects and explore the environment to discover unseen objects. Our experiments show that the proposed methods significantly outperform leading model-based reinforcement learning methods on this environment, while also yielding competitive performance on tasks from the DeepMind Control Suite.

1 INTRODUCTION

Generative models for predicting future frames of a sequence conditioned on previously observed inputs have recently achieved a lot of attention. An important use case is latent dynamics models, where future states are generated directly in a compact abstract latent space without feeding high-dimensional predicted observations back to the model. Such models have improved long-term prediction while having lower memory footprint which allows to fit larger batches on a single GPU, greatly facilitating optimization. These models have shown promising results on various tasks like video prediction (Karl et al., 2016; Kalman, 1960; Krishnan et al., 2015), model-based Reinforcement Learning (RL) (Hafner et al., 2019; 2020; 2018; Ha & Schmidhuber, 2018), and robotics (Watter et al., 2015). In visual control tasks, world models (Ha & Schmidhuber, 2018) are trained using collected interactions with an unknown environment, and behaviors are derived on sequences imagined in an abstract latent space. The recently proposed DreamerV1 (Hafner et al., 2019) and DreamerV2 (Hafner et al., 2020) solve various continuous and discrete visual control tasks from high-dimensional images.

The Dreamer agents use the Recurrent State Space Model (RSSM) (Hafner et al., 2018) comprising of an Recurrent Neural Network (RNN) to update a latent representation at each time step. Training an RNNs for long sequences is challenging as it suffers from optimization problems like vanishing gradients (Hochreiter, 1991; Bengio et al., 1994). Furthermore, many complex tasks require reliable long-term prediction of dynamics. This is true especially in partially-observable environments where only a subspace is visible to the agent, and it is usually required to accurately retain information over many time steps. To address the aforementioned issues, different ways of applying sparse updates in an RNNs have been investigated (Campos et al., 2017; Neil et al., 2016; Goyal et al., 2019), enabling

[1]École de technologie supérieure, [2]Mila, [3]MPI-IS Tübingen, [4]Université de Montréal, [5]University of Toronto, [6]Google Brain, [7]CIFAR. *Correspodence to arnavkj95@gmail.com.

a subset of state dimensions to exactly constant during the update. This alleviates the vanishing gradient problem by effectively reducing the number of sequential operations (Campos et al., 2017). Discrete gates may also improve long-term memory by avoiding the gradual change of state values introduced by repeated multiplication with continuous gate values in standard recurrent architectures.

In this work, we introduce Variational Sparse Gating (VSG), a stochastic gating mechanism for the RSSM, which sparsely updates the latent state, allowing parts of the hidden state to remain constant in an update. This sparse update prior is motivated by the observation that in the real world, many factors of variation are constant over extended periods of time. For instance, several objects in a physical simulation may be stationary until some force acts upon them. This sparse update prior might bias the model towards learning a disentangled representation with dynamic and time-independent latent factors, the latter of which can remain exactly constant over many time steps. This is useful for instance in partially-observable environments, where the agent only observes a constrained viewpoint and has to keep track of objects that are not seen for many time steps.

RSSM and similar latent dynamics models combine a stochastic with a deterministic path. While the stochastic path accounts for multiple possible future states, the deterministic path is motivated by stable retention of information over multiple time steps, facilitating gradient-based optimization (Hafner et al., 2018). In this work, we also present a variant of our VSG model with a single purely stochastic path. Having a single hidden state to capture uncertainty simplifies the model and thus we call it Simple Variational Sparse Gating (SVSG). Lastly, we developed a new 2D-Physics environment, ShapeHerd, where the task is to push shapes to a predefined goal area. In this environment, the agent gets a partial view at each step. The environment tests the agent's ability to remember states of previously observed objects and to discover new objects in the arena. Experiments were also conducted on many continuous control tasks in DeepMind Control Suite (Tassa et al., 2018) and the proposed methods achieved competitive results when compared with leading architectures. Additionally, VSG and SVSG were performing significantly better on tasks having sparse rewards.

The key contributions of this paper are summarized as follows:

- We introduce Variational Sparse Gating (VSG), a latent dynamics model with a prior for a sparsely changing latent state, enabling robust long-term predictions. We also introduce a simplified variant, Simple Variational Sparse Gating (SVSG) with a single stochastic recurrent path.
- We developed and experimented on ShapeHerd, a novel environment designed to evaluate an agent's ability to model and interact with a dynamic partially-observable setting.

2 Related Work

Latent Dynamics Models: Latent dynamics models (Bourlard & Morgan, 2012; Kalman, 1960; Bengio et al., 1999) operate directly on sequences predicted in the latent space rather than autoregressively feeding back the generated frames to the model. Recent advancements in deep learning have allowed learning expressive latent dynamics models using stochastic backpropagation (Kingma & Welling, 2013; Chung et al., 2015; Krishnan et al., 2015; Karl et al., 2016). PLaNet introduced Recurrent State Space Models (RSSM) (Hafner et al., 2018) where the latent dynamics models comprised of stochastic and deterministic components. VideoFlow (Kumar et al., 2019) predicted future values of the latent state by normalizing flows for robotic object interactions. Hierarchial latent models for video prediction were proposed in Clockwork VAEs (Saxena et al., 2021) with levels ticking at different intervals in time.

Sparsity: Neural networks have widely adopted sparsity to reduce the memory footprint of weights and activations (LeCun et al., 1990; Chen et al., 2015; Han et al., 2015). Several works have explored sparsity in RNNs. Narang et al. (2017), for instance, proposed a method for pruning RNN parameters during the initial training while keeping the performance comparable to that of a dense model. Campos et al. (2017) introduced a mechanism in RNNs that learns to skip state updates, effectively reducing the number of sequential operations on the latent state, alleviating the problem of vanishing gradients in training on long sequences. Goyal et al. (2019) presented Recurrent Independent Mechanism (RIM), an architecture that consists of separate recurrent modules which are sparsely updated using a learned attention mechanism. In contrast to RIM, the VSG architecture proposed in this work, does not use the inductive bias of a fixed modular structure. While RIM fixes the number of modules to be updated, the number of updated state variables in VSG is determined by the gating network, which is trained to satisfy a sparsity prior.

RL for Visual Control: Contemporary Deep Reinforcement Learning (DRL) methods fall into one of two categories: 1) *Model Based*- where an explicit model of the environment and its dynamics are learned (Ha & Schmidhuber, 2018; Hafner et al., 2018; 2019; 2020), and 2) *Model Free*- where a policy is learned directly from the raw observations (Srinivas et al., 2020; Kostrikov et al., 2020b; Lillicrap et al., 2015). Some recent works in Model-Free RL have focused on improving the data-efficiency on DMContol (Tassa et al., 2018) and Atari (Bellemare et al., 2013) environments. Deep Deterministic Policy Gradient (DDPG) (Lillicrap et al., 2015) combined actor-critic with insights from DQNs (Mnih et al., 2015) to learn agents for continuous action spaces. TD3 (Fujimoto et al., 2018) builds upon the DDPG algorithm and addresses the problem of overestimation bias in the value function. In particular, it utilizes clipped double Q-learning, delayed updates, and target policy smoothing. To further improve the data-efficiency, CURL (Srinivas et al., 2020) uses contrastive losses to learn discriminative representations. DrQ (Kostrikov et al., 2020a) and DrQ-v2 (Yarats et al., 2021) employed data augmentation techniques and do not use auxiliary losses or pre-training.

Model-based RL methods train agents in a two-stage process. Firstly, a dynamics model of the environment is learned and planning or behaviour learning is done on imagined trajectories. SimPLe (Kaiser et al., 2019) trains a PPO (Schulman et al., 2017) agent on the learned video generation model in pixel space. RSSMs were proposed in PlaNet (Hafner et al., 2018) to solve locomotion tasks using latent online planning. SOLAR (Zhang et al., 2019) solved robotics tasks via guided policy search in latent space. DreamerV1 (Hafner et al., 2019) learned behaviours by propagating analytic gradients of learned state values through imagined trajectories and achieved state-of-the-performance on DMControl Suite (Tassa et al., 2018). DreamerV2 (Hafner et al., 2020) further improved it by having categorical latents for the stochastic path and balancing the cross-entropy terms in the KL loss. DreamerV2 reached human-level performance on the Atari benchmark (Bellemare et al., 2013).

3 Variational Stochastic Gating

Reinforcement Learning: The visual control task can be formulated as a partially observable Markov Decision Process (POMDP) with discrete time steps $t \in [1; T]$. The agent selects an action $a_t \sim p(a_t|o_{\leq t}, a_{<t})$ to interact with the unknown environment and is provided with the next observation and a scalar reward $o_t, r_t \sim p(o_t, r_t|o_{<t}, r_{<t})$ respectively. The goal is to learn a policy that maximizes the expected discounted sum of rewards $\mathbb{E}_p(\sum_{t=1}^{T} r_t)$ over the distribution of environment and policy.

Agent: The agent has two components- the world model (3.1) and the policy (3.2). The world model learns to encode a sequence of observations and actions into a compact latent representation. In the latent space, behaviors are derived by training a policy to maximize the expected returns. For training, the initial set of episodes are collected using a random policy. As training progresses, new episodes are collected using the latest policy to further improve the world model.

3.1 World Models

World Models (Ha & Schmidhuber, 2018; Hafner et al., 2018) learn to mimic the environment using the collected experience to facilitate deriving behaviours in the abstract latent space. They have much lower memory footprint which allows learning policies with large batches in parallel. Given an abstract state of the world and an action, the model applies the learned transition dynamics to predict the resulting target state and reward. RSSMs (Hafner et al., 2018) were introduced in PlaNet where the model state was composed of two components. The deterministic transitions are implemented with an RNN, and motivated by reliable long-term information preservation, the stochastic path implements sampling from a distribution of multiple possible futures (Babaeizadeh et al., 2017). However, in many complex tasks, agents are required to have long-term memory where many objects are changing their states and the view is partial. In this work, we propose VSG, where the RNN only updates a subset of the latent states using a stochastic gating network. In contrast to some of the recently proposed latent dynamics models, the proposed method has no purely deterministic transitions. The gating mechanism selectively updates the latent state, while copying the previous values of the others. The idea of skipping state updates has been explored before, without the aspect of learning the conditional updates in Krueger et al. (2016), and without the notion of stochasticity in Campos et al. (2017); Goyal et al. (2019).

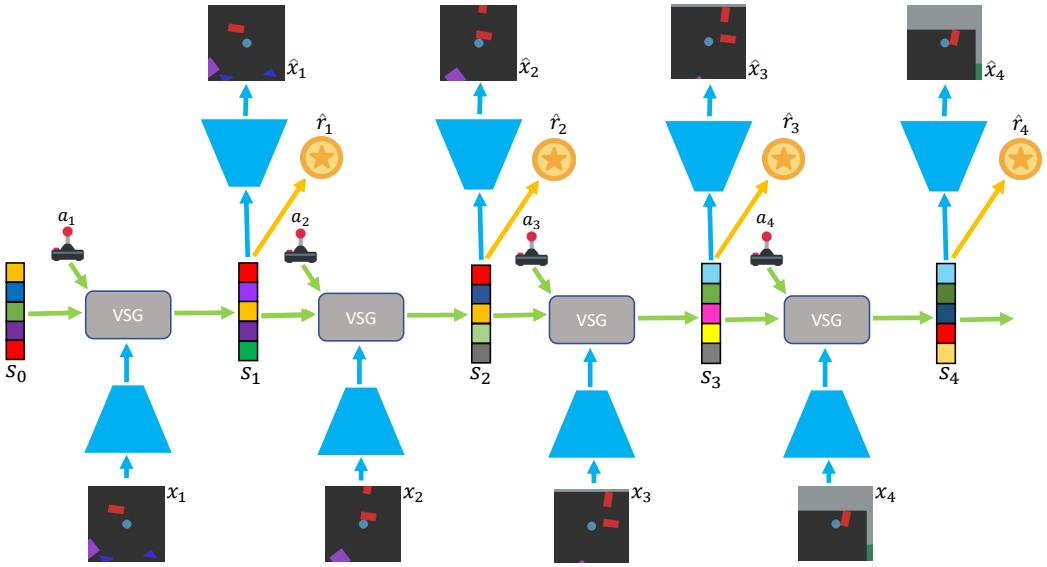


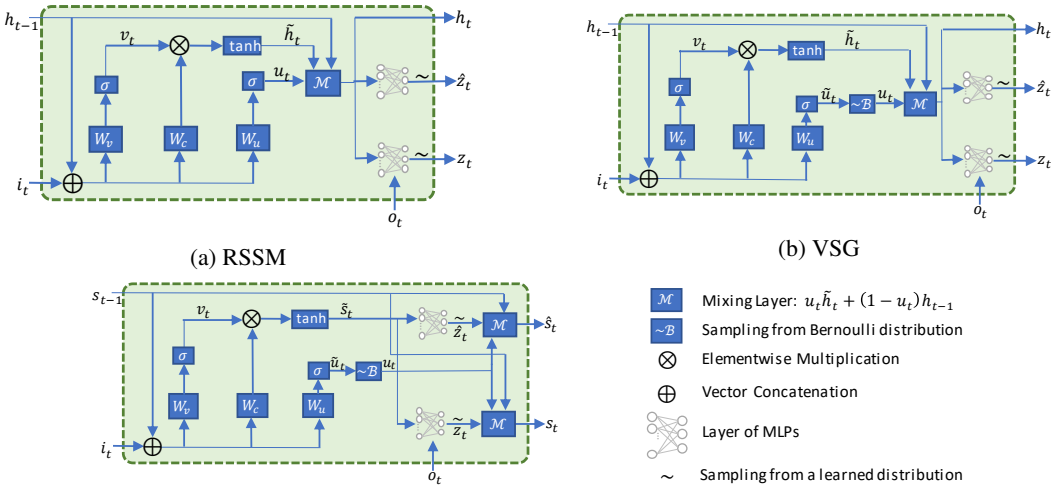
Figure 1: World Model. The images are encoded using Convolutional Neural Network (CNN) and Multi-layer Perceptron (MLP) layers. The VSG takes the previous model state s_{t-1} and returns the updated model state s_t at the next step, which is further used to reconstruct image $\hat{x}_t$ and reward $\hat{r}_t$.

Model Components: The world model comprises of an image encoder, a VSG model, and predictors for image, discount and reward. The image encoder generates representations for the observation (x_t) using CNNs. The VSG model uses a sequence of recurrent states h_t to compute two stochastic image representation states at each step, which are sampled from a distribution with learned parameters, such as from a Gaussian (Hafner et al., 2019) or a Categorical (Hafner et al., 2020) distribution. The posterior representation z_t contains information about the current image x_t and uses the encoding obtained from the image encoder o_t. The prior representation state $\hat{z}_t$ tries to predict the posterior without using the image observation and is used for planning in the latent horizon. Only the prior states are used while planning, allowing the model to predict ahead in the compact latent space without having to observe or imagine the corresponding images. This results in a low memory footprint and fast predictions of thousands of imagined trajectories in parallel on a single GPU. The concatenation of both the recurrent state and the image representation gives the compact model state ($s_t = [h_t, z_t]$). The posterior model state is further used to reconstruct the original image x_t, and predict the reward r_t and discount factor γ_t. The discount factor helps to predict the probability that an episode will end and is useful while planning in the latent space. The model components are as follows:

$$\text{VSG} \begin{cases} \text{Recurrent model:} & h_t = f_\phi(h_{t-1}, z_{t-1}, a_{t-1}) \\ \text{Representation model:} & z_t \sim q_\phi(z_t \mid h_t, x_t) \\ \text{Transition predictor:} & \hat{z}_t \sim p_\phi(\hat{z}_t \mid h_t) \\ \text{Image predictor:} & \hat{x}_t \sim p_\phi(\hat{x}_t \mid h_t, z_t) \\ \text{Reward predictor:} & \hat{r}_t \sim p_\phi(\hat{r}_t \mid h_t, z_t) \\ \text{Discount predictor:} & \hat{\gamma}_t \sim p_\phi(\hat{\gamma}_t \mid h_t, z_t) \end{cases} \tag{1}$$

Neural Networks: The representational model gives the posterior image representation state conditioned on the image encoding and recurrent state. The image encoding o_t is obtained by passing the image x_t through CNN (LeCun et al., 1989) and MLP layers. In VSG, we propose to modify the Gated Recurrent Unit (GRU) to sparsely update the recurrent state at each step. The model state, which is a concatenation of the recurrent state and the image representation is passed through several layers of an MLP to predict the discount and reward, and a transposed CNN to reconstruct the original image. The Exponential Linear Unit (ELU) activation function is used for training all the components in the model (Clevert et al., 2015).

Sparse Gating: In this work, we propose a novel gated recurrent model that sparsely updates the hidden state at each step. The input i_t to the recurrent model contains information about the action



(a) RSSM (b) VSG

(c) SVSG

Figure 2: Architectures of RSSM, VSG, and SVSG. Input i_t encodes the information about the taken action at each step. (a) RSSM uses a GRU (Cho et al., 2014) for updating the recurrent path h_t and uses it to obtain the prior $\hat{z}_t$ and posterior z_t image representation states. (b) VSG has a Bernoulli sampling based gating mechanism in the update gate u_t to sparsely update the hidden state at each step. The model state s_t for RSSM and VSG is obtained by concatenating recurrent and image representation paths. (c) SVSG has a single stochastic path to capture both the image representation and the recurrent dynamics, and sparsely updates the model state at each step.

and is obtained using the previous representation state z_{t-1} and action a_t. Similar to GRU (Cho et al., 2014), there are two gates- reset and update. The reset gate v_t decides the extent of information flow from the previous hidden state and inputs, and the update gate u_t decides which parts of the latent space will be updated. Specifically, the update gate is sampled from a Bernoulli distribution and takes only binary values ($u_t \in \{0, 1\}$)- selecting whether the recurrent state will be updated or copied from previous time step. Straight-through estimators (Bengio et al., 2013) were employed for propagating gradients backwards from the Bernoulli distribution for training. The update equations are as follows:

$$
\begin{aligned}
v_t &= \sigma(W_v^T[h_{t-1}, i_t] + b_v) \\
\tilde{u}_t &= \sigma(W_u^T[h_{t-1}, i_t] + b_u) \\
\tilde{h}_t &= tanh(v_t * (W_c^T[h_{t-1}, i_t] + b_c)) \\
u_t &\sim \textit{Bernoulli}(\tilde{u}_t) \\
h_t &= u_t \odot \tilde{h}_t + (1 - u_t) \odot h_{t-1},
\end{aligned}
\tag{2}
$$

where $\odot$ denotes element-wise multiplication. To control the sparsity of the hidden state updates, KL divergence between the probability of the update gate and the prior probability was added to the loss function. The prior probability κ is a tunable hyperparameter and was set to 0.4 in our experiments.

Loss function: The predictors for image and reward produce Gaussian distributions with unit variance, whereas the discount predictor predicts a Bernoulli likelihood. The image representation encoding is sampled from a Gaussian (Hafner et al., 2019) or a Categorical (Hafner et al., 2020) distribution obtained from the representation model and the transition predictor for posterior and prior states, respectively. The distributions are trained to maximize the likelihood of targets. Moreover, there is an entropy regularizer to make the model more robust to different model states during training. In addition, there is a KL Divergence term between the prior and posterior image distribution and similar to DreamerV2 (Hafner et al., 2020), we have also employed KL balancing. There is also a sparsity loss to regularize the number of updates in the hidden states at each time step. All the

components are optimized jointly using the loss function given by:

$$\mathcal{L}(\phi) \doteq \mathrm{E}_{q_\phi(z_{1:T} \mid a_{1:T}, x_{1:T})} \Big[\sum_{t=1}^{T} \underbrace{-\ln p_\phi(x_t \mid h_t, z_t)}_{\text{image log loss}} \underbrace{-\ln p_\phi(r_t \mid h_t, z_t)}_{\text{reward log loss}} \underbrace{-\ln p_\phi(\gamma_t \mid h_t, z_t)}_{\text{discount log loss}} \underbrace{+\beta \,\mathrm{KL}\big[q_\phi(z_t \mid h_t, x_t) \,\big\|\, p_\phi(z_t \mid h_t)\big]}_{\text{KL loss}} \underbrace{+\alpha \,\mathrm{KL}\big[\tilde{u}_t \,\big\|\, \kappa\big]}_{\text{sparsity loss}} \Big], \quad (3)$$

where β and α are the scale for KL loss and sparsity loss, respectively.

3.2 Behavior Learning

The policy comprises of a stochastic actor and a deterministic critic to learn behaviours in the latent space. The actor learns to choose the most optimal actions conditioned on the model state ($\hat{a}_t \sim p_\psi(\hat{a}_t \mid \hat{s}_t)$). The critic estimates the discounted sum of future rewards that are beyond the planning horizon $v_\xi(\hat{s}_t \approx \mathrm{E}_{p_\phi, p_\psi} \Big[\sum_{\tau \geq t} \hat{\gamma}^{\tau - t} \hat{r}_\tau \Big]$.

Critic Loss: Temporal Difference learning is used to update the parameters of the critic. The target is estimated by combining the predicted rewards from the latent model states and value estimates from the critic. The weighted average of n-step returns (V_λ) proposed in DreamerV1 (Hafner et al., 2019) is used. The critic parameters (ξ) are optimized using the mean-squared error (MSE) between the predicted value and the λ-target over all the states in a trajectory, given by:

$$\mathcal{L}(\xi) \doteq \mathrm{E}_{p_\phi, p_\psi} \Big[\tfrac{1}{H-1} \textstyle\sum_{t=1}^{H-1} \tfrac{1}{2} \big(v_\xi(\hat{s}_t) - \mathrm{sg}(V_t^\lambda) \big)^2 \Big], \quad (4)$$

where sg denotes stopping gradients at the target while updating the critic, and H denotes the length of the planning horizon in latent space which was kept to 15 in our experiments. Furthermore, the targets are computed using a copy of the critic which is updated after every 100 gradient steps.

Actor Loss: The actor is trained to maximize the λ-return computed for training the critic. The reparameterization trick (Hafner et al., 2019; Kingma & Dhariwal, 2018; Rezende et al., 2014) was used to backpropagate gradients from the value estimate. The entropy of the actor distribution is also regularized to encourage exploration. The loss for training the actor parameters (ϕ) is given by:

$$\mathcal{L}(\psi) \doteq \mathrm{E}_{p_\phi, p_\psi} \Big[\tfrac{1}{H-1} \textstyle\sum_{t=1}^{H-1} \big(\underbrace{-\eta_d V_t^\lambda}_{\substack{\text{dynamics} \\ \text{backprop}}} \underbrace{-\eta_e \,\mathrm{H}[a_t | \hat{s}_t]}_{\text{entropy regularizer}} \big) \Big]. \quad (5)$$

The actor loss was optimized with respect to the actor parameters ψ using an Adam (Kingma & Ba, 2014) optimizer. The $\eta_d = 1.0$ and the entropy regularizer $\eta_e = 10^{-4}$ were used for training.

4 Simplified Variational Stochastic Gating

In contrast to RSSM, where the model state is composed of both deterministic and stochastic components, both components in VSG (recurrent state and image representation) are stochastic. In this work, we further introduce a simplified version of VSG, called Simple Variational Sparse Gating (SVSG) which only has one stochastic path (Figure 2c [c] shows the SVSG architecture). In contrast with RSSM and VSG, all parts of the model state in SVSG are sampled from a learned probability distribution. Furthermore, the model state s_t is sparsely updated at each step through the gating mechanism discussed in VSG. In SVSG, the posterior state s_t is conditioned on the previous state s_{t-1}, input image x_t and the action a_t taken by the agent. The prior state $\hat{s}_t$ is generated without using the image observation. The states are sampled from a diagonal Gaussian distribution with learnable mean and standard deviation. Similar to VSG, the posterior state is used to reconstruct the image, and predict the reward and discount factor. The components of SVSG are:

$$\begin{aligned}
&\text{Representation model:} && s_t \sim q_\phi(s_t \mid s_{t-1}, x_t, a_t) \\
&\text{Transition predictor:} && \hat{s}_t \sim p_\phi(\hat{s}_t \mid s_{t-1}, a_t) \\
&\text{Image predictor:} && \hat{x}_t \sim p_\phi(\hat{x}_t \mid s_t) \\
&\text{Reward predictor:} && \hat{r}_t \sim p_\phi(\hat{r}_t \mid s_t) \\
&\text{Discount predictor:} && \hat{\gamma}_t \sim p_\phi(\hat{\gamma}_t \mid s_t).
\end{aligned} \quad (6)$$

The recurrent model in VSG is modified to directly output the stochastic states at each step. The candidate state $\tilde{s}_t$ is obtained using input i_t, reset gate v_t, and previous state s_{t-1}. The update gate u_t is sampled through a Bernoulli distribution to sparsely update the latent state at each step, given by:

$$\begin{aligned} v_t &= \sigma(W_v^T[s_{t-1}, i_t] + b_v) \\ \tilde{u}_t &= \sigma(W_u^T[s_{t-1}, i_t] + b_u) \\ \tilde{s}_t &= tanh(v_t \odot (W_c^T[s_{t-1}, i_t] + b_c)) \\ u_t &\sim \textit{Bernoulli}(\tilde{u}_t) \end{aligned} \tag{7}$$

where $\odot$ denotes the element-wise multiplication. It was observed that stochastically gated recurrent update was important for the convergence of the SVSG model.

The candidate state $\tilde{s}_t$ is forwarded through MLP layers to obtain parameters for the prior and posterior distributions, respectively. Analogous to VSG, the image encoding o_t is used to get the posterior distribution, where the prior parameters are predicted without the image observation. After sampling the prior $\hat{z}_t$ and posterior z_t candidate states, the update gate sparsely modifies the previous latent state and outputs prior $\hat{s}_t$ and posterior s_t states, respectively. The equations are:

$$\begin{aligned} z_t &\sim f_{\text{post}}(\tilde{s}_t, o_t) \\ \hat{z}_t &\sim f_{\text{prior}}(\tilde{s}_t) \\ s_t &= u_t \odot z_t + (1 - u_t) \odot s_{t-1} \\ \hat{s}_t &= u_t \odot \hat{z}_t + (1 - u_t) \odot s_{t-1} \end{aligned} \tag{8}$$

where both f_{post} and f_{prior} denote distributions with learnable parameters. We did not see good performance on our tasks using a categorical distribution. We attribute this to the fact that samples from a categorical distribution are binary vectors and it's hard to perform reconstruction accurately with such latent space embeddings. Lastly, the sampled states are used to sparsely update the latent representations to obtain the prior and posterior model states, respectively.

For training the SVSG model, we replace the KL loss term in Eq. 3 with a masked KL loss that penalizes the state dimensions that were updated in the corresponding time step, i.e. those for which the corresponding element in u_t is equal to 1. We found this to be necessary, since the original, unmasked KL loss did not yield good performance, presumably due to its effect on state dimensions that were not updated.

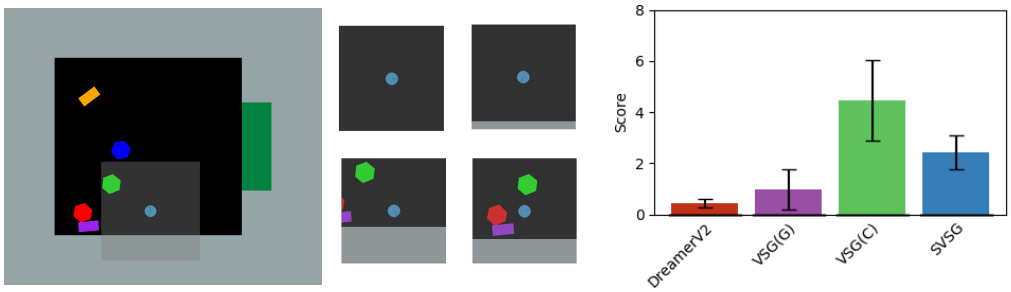
5 Results

To evaluate the efficacy of VSG and SVSG, we conducted experiments on two RL benchmarks. Section 5.1 introduces the novel ShapeHerd environment and presents the corresponding experimental evaluation. Section 5.2 describes our experiments and results on a wide variety of tasks from the well-established DeepMind Control Suite (Tassa et al., 2018).

5.1 ShapeHerd

The ShapeHerd environment is developed to test for the long term memory and exploration capabilities of RL agents. The agent is tasked to push shapes within the arena into a specified goal area. The agent receives an obfuscated view of the arena around its current position. Solving tasks with partial-observability requires agents that can explore the arena to find new shapes as well as remember the states of previously seen objects. Furthermore, the objects can collide with each other and the walls, which further requires agents to update the state of objects not present in the current view. We used the Pymunk (Blomqvist, 2007) physics simulator for developing the environment.

The environment has a circular blue agent which can move in any direction. The shapes and colors of the objects are uniformly sampled from a predefined set of 5 shapes and 5 colors. As there are 5 objects in the arena for each episode, there are $25^5 \sim 9.7M$ possible combinations of different objects. The initial positions of the agent and the objects are randomly chosen within the arena. The elasticity of the objects and the agent is set to 1.0, while the walls have an elasticity of 0.7. There is a damping factor of 0.3 applied to the velocities of all the agent and all the objects. In Fig. 3a, we show a full view of the whole arena at the beginning of an episode, and Fig. 3b presents the agent's view at different time steps. It can be observed that the agent might see none or all of the objects in its



(a) (b) (c)

Figure 3: a) Full arena of ShapeHerd environment, where the grey box around the agent is its partial view. b) Some observations within an episode of the agent's view. In general, the agent observes less than 25% of the arena. This requires the agent to retain information of previously seen objects in its latent representation. c) Experimental results on the ShapeHerd environment. VSG and SVSG significantly outperform DreamerV2 (Hafner et al., 2019). VSG(C) and VSG(G) denote the variants of VSG with Categorical and Gaussian latents, respectively.

current view and it needs to explore it environment to look for all the objects and push them into the goal area (the green area in Fig. 3a)

The agent receives images of size $64 \times 64 \times 3$ from the environment. The action space is continuous and comprises of the angle and the magnitude of force applied to the agent. For pushing an object in the goal area, the agent gets a reward of +1. Furthermore, additional reward is given when the objects are pushed into the goal earlier in the episode. The reward function is:

$$r_t = \begin{cases} 1 + max(0, 1.5 - \frac{t}{1000}) & \text{if an object is pushed into the goal} \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

where t denotes the current time step in the episode. An episode terminates once the agent pushes all the objects into the goal, or if 3000 environment steps are completed. Agents were trained for 1M environment steps with an action repeat of $R = 4$. The agents are evaluated on 20 episodes and we report the mean and the standard deviation of the sum of the episode rewards.

Fig. 3c compares the proposed methods with DreamerV2 (Hafner et al., 2019), which is a state-of-the-art model based RL agent. For VSG, we have compared with two variants having categorical and Gaussian latents, VSG(C) and VSG(G), respectively. As discussed in Sec. 4, we trained SVSG with Gaussian latents only. The proposed variants perform significantly better than DreamerV2. This indicates that utilizing a sparsity prior might help agents in learning robust representations, thereby enabling policies to quickly learn from imagined trajectories in latent space. VSG(C) performs better than other variants and achieves a mean score of **4.45**, demonstrating the ability of the learned agent to push around 3-4 objects on average. Furthermore, SVSG performs better than VSG(G), where we use Gaussian latents. We believe that having a single stochastic path better captures the uncertainty in the state which helps in efficiently exploring the arena. Qualitatively, we also observed the maneuvers taken by the agents to push the objects to the goal (please refer to the supplementary material for videos). The DreamerV2 agents were able to recognize objects and get behind them to push them, but were not found to follow a smooth trajectory, failing in most cases. The proposed methods, especially VSG(C), had comparatively much smoother trajectories and were more efficient at pushing objects into the goal area.

5.2 DeepMind Control Suite

The proposed method is evaluated on a few tasks from the DeepMind Control Suite (Tassa et al., 2018). Observations for the agents are high-dimensional images of shape $64 \times 64 \times 3$, action dimension ranges from 1 to 12, and episodes last for 1000 steps. An action repeat (Mnih et al., 2016) of $R = 2$ was used. The model was implemented using Tensorflow Probabability (Dillon et al., 2017) and trains on a single NVIDIA V100 GPU with 16GB memory in less than 6 hours. The agents were trained for 500K environment steps. Baselines include DreamerV1 (Hafner et al., 2019),

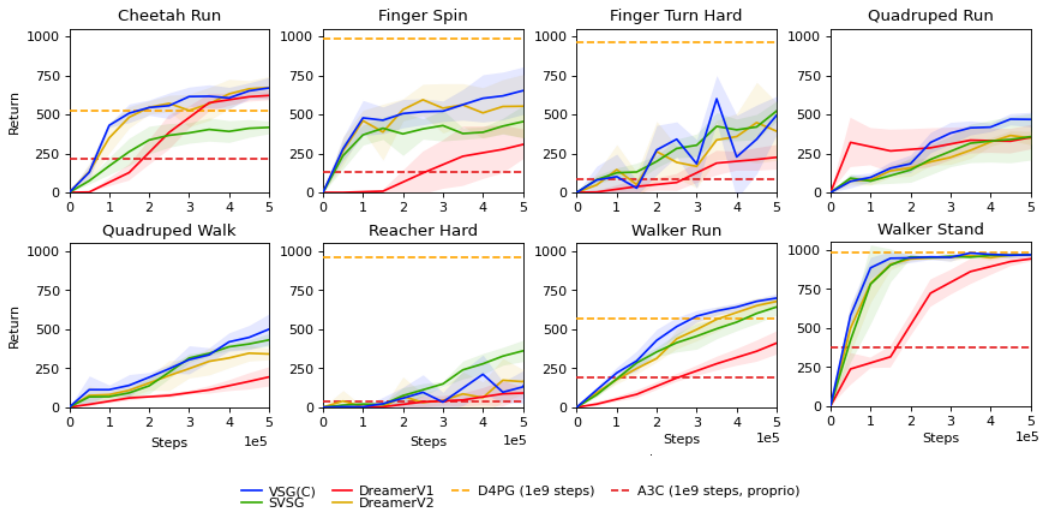


Figure 4: Comparison of VSG and SVSG with DreamerV1 (Hafner et al., 2019) and DreamerV2 (Hafner et al., 2020) on the DeepMind Control Suite. VSG shows faster learning as demonstrated by the evaluation curves. Even with a single stochastic path, SVSG achieves performance competitive with strong model-based baselines that use a combination of stochastic and deterministic vectors in the latent state.

DreamerV2 (Hafner et al., 2020), D4PG (Barth-Maron et al., 2018), and A3C (Mnih et al., 2016). Except A3C, all baselines learn policies from high dimensional pixel inputs. DreamerV2 was trained using the implementation provided by the authors. For other baselines, the metrics provided by the respective authors were used for comparison. For all the methods, we report returns averaged over 5 seeds.

Figure 4 presents a comparison of VSG and SVSG with the aforementioned baselines on 8 tasks from DMControl Suite (Tassa et al., 2018). For evaluating the performance of VSG, we replaced RSSM with VSG in DreamerV2 (Hafner et al., 2020). We call it VSG(C) as categorical latent was used in DreamerV2. It can be observed that on most tasks, VSG(C) performs comparable to or better than DreamerV2 (Hafner et al., 2020), which is a state-of-the-art model based RL agent. VSG significantly outperforms DreamerV2 on Quadruped tasks where the rewards signals are sparse. This demonstrates that better gradients were propagated back due to sparsity, leading to faster convergence of the model. Furthermore, it can be observed that SVSG, though having a single component in model state, has similar performance as DreamerV2. In addition, SVSG significantly outperforms DreamerV1, which used Gaussian distribution based image representation state, on many tasks .

6 Conclusion

In this work, we introduce VSGand SVSG, two latent dynamics models leveraging stochastic sparse state updates. We show that the proposed architectures can solve the newly proposed ShapeHerd task, a challenging partially-observable environment, which requires exploration and remembering long-term dependencies. The proposed models also achieve comparable performance to DreamerV2 on many tasks of the DeepMind Control Suite. The sparse update prior can facilitate more efficient behaviors in tasks requiring long-horizon planning. In the current implementation of VSG, the latent space does not exhibit disentanglement. To address this, VSG can be combined with other recurrent architectures, such as the independent modules of RIM (Goyal et al., 2019).

Acknowledgements

The authors would like to thank David Meger and Ankesh Anand for their valuable feedback and discussions. The authors are also grateful to CIFAR for funding and Compute Canada for computing resources.

REFERENCES

Mohammad Babaeizadeh, Chelsea Finn, Dumitru Erhan, Roy H Campbell, and Sergey Levine. Stochastic variational video prediction. *arXiv preprint arXiv:1710.11252*, 2017.

Gabriel Barth-Maron, Matthew W Hoffman, David Budden, Will Dabney, Dan Horgan, Alistair Muldal, Nicolas Heess, and Timothy Lillicrap. Distributed distributional deterministic policy gradients. *arXiv preprint arXiv:1804.08617*, 2018.

Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47: 253–279, 2013.

Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

Yoshua Bengio et al. Markovian models for sequential data. *Neural computing surveys*, 2(199): 129–162, 1999.

Victor Blomqvist. Pymunk: A easy-to-use pythonic rigid body 2d physics library (version 6.2.0), 2007. URL https://www.pymunk.org.

Herve A Bourlard and Nelson Morgan. *Connectionist speech recognition: a hybrid approach*, volume 247. Springer Science & Business Media, 2012.

Víctor Campos, Brendan Jou, Xavier Giró-i Nieto, Jordi Torres, and Shih-Fu Chang. Skip rnn: Learning to skip state updates in recurrent neural networks. *arXiv preprint arXiv:1708.06834*, 2017.

Wenlin Chen, James Wilson, Stephen Tyree, Kilian Weinberger, and Yixin Chen. Compressing neural networks with the hashing trick. In *International conference on machine learning*, pp. 2285–2294. PMLR, 2015.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Junyoung Chung, Kyle Kastner, Laurent Dinh, Kratarth Goel, Aaron C Courville, and Yoshua Bengio. A recurrent latent variable model for sequential data. In *Advances in neural information processing systems*, pp. 2980–2988, 2015.

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

Joshua V Dillon, Ian Langmore, Dustin Tran, Eugene Brevdo, Srinivas Vasudevan, Dave Moore, Brian Patton, Alex Alemi, Matt Hoffman, and Rif A Saurous. Tensorflow distributions. *arXiv preprint arXiv:1711.10604*, 2017.

Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*, 2018.

Anirudh Goyal, Alex Lamb, Jordan Hoffmann, Shagun Sodhani, Sergey Levine, Yoshua Bengio, and Bernhard Schölkopf. Recurrent independent mechanisms. *arXiv preprint arXiv:1909.10893*, 2019.

David Ha and Jürgen Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2018.

Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. *arXiv preprint arXiv:1811.04551*, 2018.

Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*, 2019.

Danijar Hafner, Timothy Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models. *arXiv preprint arXiv:2010.02193*, 2020.

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

Sepp Hochreiter. Untersuchungen zu dynamischen neuronalen netzen. *Diploma, Technische Universität München*, 91(1), 1991.

Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, et al. Model-based reinforcement learning for atari. *arXiv preprint arXiv:1903.00374*, 2019.

Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, 82(1):35–45, 1960.

Maximilian Karl, Maximilian Soelch, Justin Bayer, and Patrick van der Smagt. Deep variational bayes filters: Unsupervised learning of state space models from raw data. *arXiv preprint arXiv:1605.06432*, 2016.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Diederik P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. *arXiv preprint arXiv:1807.03039*, 2018.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Ilya Kostrikov, Denis Yarats, and Rob Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. *arXiv preprint arXiv:2004.13649*, 2020a.

Ilya Kostrikov, Denis Yarats, and Rob Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. *arXiv preprint arXiv:2004.13649*, 2020b.

Rahul G Krishnan, Uri Shalit, and David Sontag. Deep kalman filters. *arXiv preprint arXiv:1511.05121*, 2015.

David Krueger, Tegan Maharaj, János Kramár, Mohammad Pezeshki, Nicolas Ballas, Nan Rosemary Ke, Anirudh Goyal, Yoshua Bengio, Aaron Courville, and Chris Pal. Zoneout: Regularizing rnns by randomly preserving hidden activations. *arXiv preprint arXiv:1606.01305*, 2016.

Manoj Kumar, Mohammad Babaeizadeh, Dumitru Erhan, Chelsea Finn, Sergey Levine, Laurent Dinh, and Durk Kingma. Videoflow: A flow-based generative model for video. *arXiv preprint arXiv:1903.01434*, 2(5), 2019.

Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In *Advances in neural information processing systems*, pp. 598–605, 1990.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pp. 1928–1937, 2016.

Sharan Narang, Erich Elsen, Gregory Diamos, and Shubho Sengupta. Exploring sparsity in recurrent neural networks. *arXiv preprint arXiv:1704.05119*, 2017.

Daniel Neil, Michael Pfeiffer, and Shih-Chii Liu. Phased lstm: Accelerating recurrent network training for long or event-based sequences. *arXiv preprint arXiv:1610.09513*, 2016.

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*, 2014.

Vaibhav Saxena, Jimmy Ba, and Danijar Hafner. Clockwork variational autoencoders. *arXiv preprint arXiv:2102.09532*, 2021.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Aravind Srinivas, Michael Laskin, and Pieter Abbeel. Curl: Contrastive unsupervised representations for reinforcement learning. *arXiv preprint arXiv:2004.04136*, 2020.

Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

Manuel Watter, Jost Springenberg, Joschka Boedecker, and Martin Riedmiller. Embed to control: A locally linear latent dynamics model for control from raw images. In *Advances in neural information processing systems*, pp. 2746–2754, 2015.

Denis Yarats, Rob Fergus, Alessandro Lazaric, and Lerrel Pinto. Mastering visual continuous control: Improved data-augmented reinforcement learning. *arXiv preprint arXiv:2107.09645*, 2021.

Marvin Zhang, Sharad Vikram, Laura Smith, Pieter Abbeel, Matthew Johnson, and Sergey Levine. Solar: deep structured representations for model-based reinforcement learning. In *International Conference on Machine Learning*, 2019.